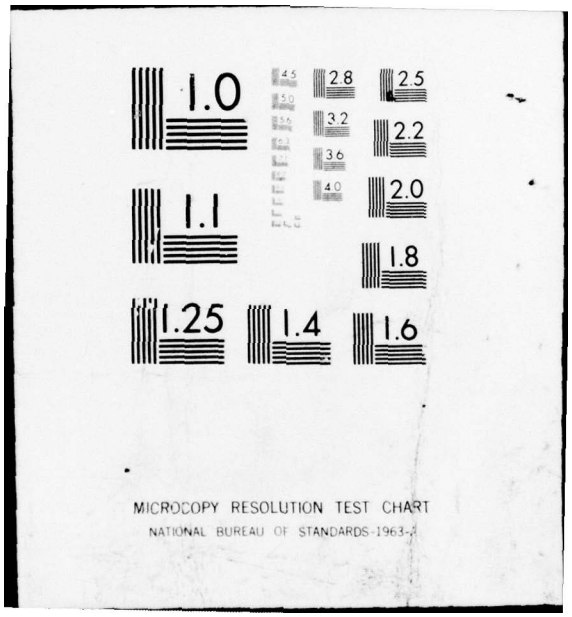
AD-A052 022 STANFORD UNIV CALIF DIGITAL SYSTEMS LAB F/G 9/2
EMMY/360 CROSS ASSEMBLER.(U)
DEC 75 T S HEDGES DAHC04-76-G-0001
UNCLASSIFIED DSL-TN-74 ARO-12958.4-M NL

1 OF 1
AD A052 022

END
DATE FILMED
5-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(18) ARO 12958.4-M

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Technical Note No. 74 12958.4-M
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: DSL-TN-74
4. TITLE (and Subtitle): EMMY/360 CROSS ASSEMBLER.
5. TYPE OF REPORT & PERIOD COVERED: Technical Note
6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Thomas S. Hedges
8. CONTRACT OR GRANT NUMBER(s): DAHC-04-76-G-0001new
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Digital Systems Laboratory, Stanford Electronics Laboratories, Stanford University, Stanford, CA 94305
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 13p.
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Research Office-Durham
12. REPORT DATE: Dec 1975
13. NUMBER OF PAGES: 10
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
15. SECURITY CLASS. (of this report)
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report): "Approved for public release; distribution unlimited."
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
18. SUPPLEMENTARY NOTES: The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A cross assembler and simulator for the EMMY microprogrammable processor has now been developed and will shortly be generally available for EMMY Lab users. The program is written in ALGOL W and presently exists on the Campus 360/67, however, in the future it should also be available at SLAC or the SCIP 168.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408071

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

EMMY/360 CROSS ASSEMBLER

by

Thomas S. Hedges

December 1975

Technical Note No. 74

Digital Systems Laboratory

Stanford Electronics Laboratories

Stanford University

Stanford, California

The work herein was supported in part by the Army Research Office-Durham under contract DAHC 04-76-G-0001.

Digital Systems Laboratory

Stanford Electronics Laboratories

Technical Note No. 74

December 1975

EMMY/360 CROSS ASSEMBLER

by

Thomas S. Hedges

ABSTRACT

A cross assembler and simulator for the EMMY micro-programmable processor has now been developed and will shortly be generally available for EMMY Lab users. The program is written in ALGOL W and presently exists on the Campus 360/67, however, in the future it should also be available at SLAC or the SCIP 168.

The work herein was supported in part by the Army Research Office-Durham under contract DAHC 04-76-G-0001.

I. THE CROSS ASSEMBLER LANGUAGE

A. CHARACTER SET: The cross assembler accepts EBCDIC characters, including ASCII: '[',']','{','}'.

B. IDENTIFIERS: Identifiers may be from one to eight (8) characters. Characters beyond the eighth will be ignored by the assembler. Identifiers conform to the following:

1. First character must be ALPHA(BETIC), upper or lower case; or the dollar sign '$'.

2. The second and following characters, if any, may be ALPHA, NUMERIC ('0'-'9'), the dollar sign, or the underscore ('_').

C. RESERVED IDENTIFIERS: Certain identifiers are reserved for special use of the assembler and are not available to the programmer as ordinary identifiers. The reserved identifiers are all upper case alphabetic characters.

D. COMMENTS: Comments may be included on any assembler statement by prefacing the comment with a period ('.') or vertical bar ('|'). A comment may be the only item on a line, and totally blank lines may be included in the input.

E. STATEMENT FORMAT: Statements are coded entirely within card columns 1-72 and no continuation is allowed. The contents of columns 73-80 is printed on the listing but otherwise ignored.

[<LABEL ID>:] [<T-STATEMENT>] [;<A-STATEMENT>[;<UPDATE PTR>]]

Machine code statements follow the basic form given above, with numerous minor variations, of course. Blanks, beyond a single one used to delimit other quantities, are ignored and coding is free form within a statement.

If no a-statement is coded then it and the semicolon preceding are omitted, and this also precludes coding any <UPDATE PTR>. Likewise if the t-statement is not coded, then it is omitted with the a- statement beginning as shown with a semicolon.

One or more label identifiers may be attached to a statement by coding each before other items on a given line, and following each LABEL I.D. with a colon (':'). If a LABEL(s) is left 'hanging', (i.e. coded on a line containing no machine statement)

then the label is assigned the current value of the location counter.

F. IDENTIFIER TYPES: The assembler defines different types of identifiers as follows:

TYPE	DESCRIPTION	RANGE OF VALUES
ABS	ABSOLUTE NUMERIC	$-2^{31} \leq n \leq 2^{31} - 1$
SYMB	SYMBOLIC LABEL	$0 \leq n \leq 2^{12} - 1$
REG	HARDWARE REGISTER	R0.R1.R2.R3.R4.R5.R6.R7
MASK	CONDITIONAL MASK	ALL 2^{11} of $2^{8} \cdot \binom{0}{1} \cdot \binom{0}{1} \cdot \binom{0}{1}$

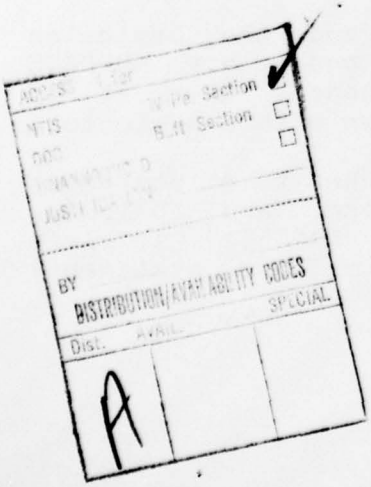
G. LITERALS: The assembler accepts literal constants coded in any of the forms below. All literal constants are like an identifier type ABS. except * which is SYMB.

>DECIMAL	nnnnnn	($\leq 2^{31} - 1$)
>HEX	X'nnnnnnnn'	($\leq$ 8 HEX DIGITS)
>OCTAL	O'nnn-n'	($\leq 2^{31} - 1$)
>BINARY	B'$\binom{0}{1}\binom{0}{1}\cdots\binom{0}{1}$'	($\leq$ 32 BINARY DIGITS)
>CHARACTER	C'cccc'	($\leq$ 4 CHARACTERS)

(NOTE: Character constants are converted to ASCII and packed right justified in 8 bit fields. No parity bit is given so the high bit of all bytes is zero.)

>LOCATION COUNTER * [TYPE IS SYMB]

(NOTE: The * gives the location counter value or the address of the current instruction being processed.)



H. EXPRESSIONS: Literals and identifiers of types ABS and SYMB may appear in expressions as follows (and only as below):

UNARY -:	--ABS	NEGATE
UNARY +:	++ABS	NULL (NO ACTION)
BINARY -:	ABS_1 -- ABS_2	YIELDS ABS RESULT
	$SYMB_1$ -- $SYMB_2$	YIELDS ABS RESULT
BINARY +:	ABS_1 ++ ABS_2	YIELDS ABS RESULT
	ABS ++ SYMB	YIELDS SYMB RESULT
	$SYMB_1$ ++ $SYMB_2$	ERROR

Multiple operators are evaluated from left to right. No parentheses may be used.

I. PSEUDO-OPS:

1. DC - DEFINE CONSTANT

DC <EXPRESSION>

This statement reserves one word of storage. The statement may have a label, whose value would be the address of the constant. The <EXPRESSION> may be ABS or SYMB.

2. BLK - BLOCK

BLK <ABS-EXPR>

Reserve <ABS-EXPR> words of storage. The label, if coded, is the address of the first word. The expression must be ABS.

3. ORG - ORIGIN LOCATION COUNTER

ORG <EXPRESSION>

Begin assembling code (starting with next statement) at location <EXPRESSION>, which may be ABS or SYMB. It is poor form to label an ORG statement although the assembler probably would allow it.

4. EQU EQUATE SYMBOL

<IDENTIFIER> EQU { <REG> | <EXPRESSION> | <MASK> | <IDENTIFIER> }

NOTE: MASKS are defined using 'MASK' function as follows:

MASK(<EIGHTBITS>.<NOT>.<ZERO>.<CODES>)

The four fields are all ABS and <EIGHTBITS> is 0≤ ≤255 while the other fields are 0 or 1 only.

<EIGHTBITS> IS THE TEST MASK
<NOT> ≅ 1 . INVERTED SENSE
<ZERO>≅1 . TEST INVERTED (FOR ZERO)
<CODES>≅1. TEST INDICATOR CODES
(INSTEAD OF CONDITION CODES)

The EQU <IDENTIFIER> must not have been used as a label identifier nor may it appear at the left in another EQU. The <IDENTIFIER> is given the type and value of the quantity on the right of the EQU. A register. mask define. expression. or identifier may appear on the right in the EQU. A restriction exists that any identifier appearing at the right. either alone or in an expression. must have been given its value earlier in the program.

5. END

END [<LABEL>]

The END PSUEDO-OP marks the last physical statement of the program. The optional <LABEL> specifies a point to transfer control to begin execution.

J. T - STATEMENT

1. ARITHMETIC

To store result & set codes:

$$\text{Raf} := \text{Raf} \begin{Bmatrix} + \\ +c+ \\ - \\ -b- \end{Bmatrix} \begin{Bmatrix} \text{Rbf} \\ \\ \text{LITERAL} \end{Bmatrix} \qquad \text{*1*}$$

To only set codes:

$$\text{Raf} \begin{Bmatrix} + \\ +c+ \\ - \\ -b- \end{Bmatrix} \begin{Bmatrix} \text{Rbf} \\ \\ \text{LITERAL} \end{Bmatrix} \qquad \text{*1*}$$

2. LOGICAL

$$\text{Raf} := \begin{Bmatrix} 0 \text{ or } -1 \\ \text{LITERAL or LABEL} \\ \text{NOT <OP2>} \\ \text{Raf <LOG> <OP2>} \\ \text{Rbf} \end{Bmatrix} \qquad \begin{matrix} \text{*2*} \\ \text{*2*} \end{matrix}$$

'-' MAY BE USED FOR 'NOT'
<OP2>: Rbf or LITERAL *2*
<LOG>: AND or OR or NAND or
NOR or XOR or XNOR

NOTE: Raf := Rbf DOES A 'LTR'. THAT IS
LOADS AND SET CONDITION CODES

---

1 LITERAL is assembled short if 0 < LIT < 7 and an A-STATEMENT is coded, otherwise it is long.

2 LITERAL or LABEL with value 0 or -1 are assembled short. other values generate a long literal form.

3. SHIFT.ROTATE

SINGLE:

Raf <S-OP> {Rbf / LITERAL} *1*

DOUBLE:

Raf, Raf⊕1 <S-OP> {Rbf / LITERAL} *1*

<S-OP>: << LEFT LOGICAL
<@ LEFT ROTATE
<< RIGHT LOGICAL
@< RIGHT ARITHMETIC

4. EXTENDED

TRANSFER

Raf = Rbf [NOTE: CONDITION CODES NOT SET]

DIVIDE STEP

DIV (Raf,Rbf)

MULTIPLY STEP

MUS (Raf,Rbf)

EXCESS SIX

XS6 (Raf,Rbf)

---

1 LITERAL is assembled short if 0< LIT <7 and an A-STATEMENT is coded. otherwise it is long.

5. EXTRACT/INSERT

Raf (<LIST>) = Rbf (<n_1>:<n_0>) [{CLEAR / INSERT}]

<LIST> is one or more of the following separated by commas

a. <i_1> : <i_0> position i_0 through i_1 *1*

b. <i_2> position i_2

Rbf is rotated assuming the <n_0> of Rbf will be matching the last i argument (furtherest to right) of <LIST> for Raf.

If 'CLEAR' is coded an EXTRACT is done otherwise an INSERT. The ACTION is to assign the bit field given by <n_1> : <n_0> in Rbf to Raf into those bits given by <LIST> and leave the the others the same (or clear them if 'CLEAR').

6. CONDITIONAL

([NOT] <MASK> => [;] <A-STATEMENT>)

The entire statement is surrounded by a set of parentheses. including the <A-STATEMENT>. The <MASK> must be either the 'MASK(i ,i ,i ,i)' or an IDENTIFIER with TYPE MASK. The test may be inverted by specifying 'NOT'. NORMAL if the test is satisfied the <A-STATEMENT> is EXECUTED, otherwise it is skipped.

---

1 If $i_0 > i_1$ then positions 31 to i and i to 0 are selected (it wraps around). Remember not all possible masks can be represented in the 18 bit literal field.

K. A-STATEMENT

1. STORE REGISTER

M(<EXPRESSION>) = Rcf

2. LOAD REGISTER

Rcf = M(<EXPRESSION>)

3. LOAD IMMEDIATE

Rcf = <EXPRESSION>

[this includes <LABEL>'s and type SYMB]

4. INDIRECT ACCESS

M(Rcf) = X(Rdf)
Rcf = X(Rdf)
X(Rcf) = Rdf
X(Rcf) = M(Rdf)
Rcf = M(Rdf)
M(Rcf) = Rdf
Rcf = Rdf

NOTE: To do POINTER UPDATE include

Either

; Rcf = Rcf {±} LIT or
; Rdf = Rdf {±} LIT or
; Rcf = Rcf {±} LIT ; Rdf = Rdf {±} LIT

5. POINTER-MOD AND LOOP

INC Rcf
DEC Rcf
Rcf = Rcf{ }Rdf

NOTE: To do CONDITIONAL LOOP include

$$\left(\left[[\text{NOT}] \begin{Bmatrix} > \\ < \\ = \end{Bmatrix} 0 \right] \left[=> \right] \begin{Bmatrix} \text{R0}\{\ \}\text{LIT} \\ \text{<LABEL>} \end{Bmatrix} \right)$$

If <LABEL> is used must be equivalent to R0 ± LIT, where $-8 \leq \text{LIT} \leq 7$

6. CONDITIONAL BRANCH

([NOT] <MASK> [=>] { RO{$\pm$}Lit)
                   { <LABEL>   }

Branch part is a above. <MASK> the same as for condition T-STATEMENT.

APPENDIX

A. PREDEFINED MASKS

ZERO	
NEGATIVE	
POSITIVE	
OVERFLOW	
CARRY	CARRY BIT = 1
HIGH	HIGH BIT = 1
LOW	LOW BIT = 1
SAME	ALL BITS 0 or 1
ODD	PARITY IS ODD
BUSY	CPU BUS ACCESS IS BUSY

B. RESERVED WORDS

R0	INSERT
R1	CLEAR
R2	
R3	MUS
R4	DIV
R5	XS6
R6	
R7	DEC
	INC
AND	
OR	ZERO
NAND	POSITIVE
NOR	NEGATIVE
XOR	OVERFLOW
XNOR	CARRY
NOT	HIGH
	LOW
MASK	SAME
EQU	ODD
BLK	BUSY
END	
DC	
ORG	
X+	
M+	

+ only when directly followed by '('